

## **Pattern Based Reengineering**

Michael A. Beedle Ph. D.  
Principal

Framework Technologies Inc.  
1901 North Roselle  
Schaumburg, IL 60195-3186  
Voice: (847) 490-7110  
Direct: (312) 218-6562  
Internet: beedlem@ix.netcom.com  
<http://www.netcom.com/~beedlem/>

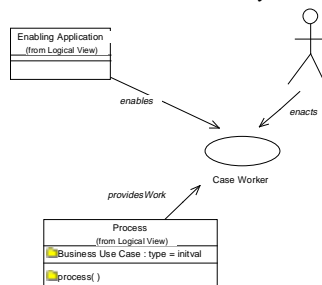
## 1. Introduction

Ivar Jacobson [Jacobson95] and David Taylor [Taylor95] have published books on business engineering using objects. Other authors, organizations and tool manufacturer's practice techniques to do business modeling and reengineering with objects, for example Gemini Consulting's **Construct**, Andersen Consulting's **Eagle Model**, the Gensym Approach through their ReThink tool [Yourdon95], SES Software's **Business Architect**, Platinum's Paradigm Plus and Rational **ROSE** to name a few. Process modeling methods based on objects offer an advantage to other traditional process modeling techniques because they facilitate the communication between business and technical people; allow the "reification" or "objectification" of processes; and provide a facilitated process to find "business objects". In fact the OMG published last year an architecture for business objects [OMG-BAA95].

I published an article in Object Magazine in 1995 with the title "Object Based Reengineering", Object Magazine, March-April 1995 which hinted at "reengineering patterns". In this presentation I include these "reengineering" patterns explicitly and added them to an extended version of OBR which I now call "Pattern Based Reengineering". Why patterns? My argument is simple. Reengineering with use cases and objects, though it provides with a number of clear advantages to other modeling languages, does not necessarily lead to better business designs and implementations. However, reengineering using patterns will, because these patterns are proven BPR solutions implementing business architecture constructs. Patterns also provide with a level of indirection that is more appealing to business people i.e. they don't have to understand object models to use patterns.

PBR is an iterative and incremental business engineering method that is based on business use cases, is architecture centric and provides a pattern language to conduct reengineering. The patterns in the language are in chronological order spanning Strategic Assessment, Business Analysis, Business Design, Business Evolution, Business Maintenance (See the "lite" version of PattBPR at the end of this paper); but the pattern language also has depth into the architectural layers: Business Organization, Software Development Organization, System Architecture and Enabling Applications. The style of the method is borrowed from Grady Booch life-cycle. Jim Coplien [Coplien95] has documented the patterns of architecture driven, iterative and incremental life-cycles. In addition PBR identifies the dependencies of the implementation of business processes from the system architecture using use case maps [Buhr96], and architectural dependency metrics across class categories published by Robert Martin [Martin95]. This order dependency in the comprehensive system is very important to evolve the system in an iterative and incremental fashion. **PBR** uses the most widely used and fast growing object modeling notation UML to do business modeling [UML96] because it is one of the few object oriented methods that includes the reification of patterns. This reification simply means "objectifying" a pattern so that it can be used directly in the business or system designs. For example, figure 1 shows how a Case Worker is implemented using UML.

*UML is necessary because it allows **patterns** to be used as parts of the architecture.*



**Figure 1. Example of pattern reification in UML [UML96].**

What is a pattern? Alexander [Alexander78 ](CH 14. Pg. 247) describes it as:

*Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.*

The paradigm that Alexander proposes to build things is based on three concepts: The Quality, The Gate and The Timeless Way. The Quality is created when the attributes in the design makes that design "live". That is, designs that are flexible, extensible, adaptable, reusable and have other qualities of living things; except of course self reproduction and metabolism. The Gate is the CPL (Common Pattern Language), which is the universal network of patterns and pattern relationships dedicated to a domain. A pattern language for a specific design is chosen by the designer from the Common Pattern Language; however, this chosen language needs to be a morphologically and functional complete. The value of single patterns cannot be underestimated, without them pattern languages would not exist. Pattern Languages are applied using The Way. That is we apply one pattern at a time, "differentiating space" to successively evolve an initial architecture and unfold it into a "live design", or said it in Alexander terms, a design with The Quality. Combining design patterns such as POSA, GOF, Schmidt's communications patterns, PLOP 1,2,3.. N proceedings, CACM and other published patterns will allow us to eventually have a CPL (Common Pattern Language) for OOD. Combining Organizational Pattern Languages, such as Coplien's and other PLOP 1,2,3 organization patterns will allow us to have a CPL to design organizations that develop OO software. The patterns presented here were found in successful reengineering projects. Many books, articles, magazines, periodicals and web pages were researched to find this information. Most influential was the work presented by Hammer and Champy "Reengineering the Corporation" [Hammer93]. PBR will be published in my upcoming book by SIGS "Reengineering the Application Development Process", this paper presents a summary of it. A paper with the complete pattern language PattBPR will also be submitted to PLOP 97.

## 2. Enterprise Models

The typical organization has a vastly diversified collection of organizations, policies, processes, systems, values and beliefs; also, the IS Organization has a large collection of diversified applications, networks, organizations, processes, projects, systems, and technologies. An enterprise model is constructed mapping the components of the Organization, its IS Organization and its System Architecture.

### 2.1 Uniqueness

In the typical organization the components of the Enterprise Model do not necessarily form a coherent whole. In fact, most of the corporate environments don't keep current enterprise models and the ones that do, do not enforce a systematic control over its evolution. In some organizations multiple incoherent models are created by the accountants, business analysts, software developers, operations staff and strategic planners.

*There must be ONE comprehensive congruent enterprise model for the Organization, its IS Organization and its System Architecture.*

### 2.2 Importance

There has been a lot discussions regarding whether you should create a enterprise model of an existing organization [Jacobson95]. However, it is important to be convinced that this is an important activity.

*It is important to know how your organization operates today because most likely you will be changing it incrementally. Keeping an ongoing model for the organization is essential to*

*manage change. In fact this model should be available through a configuration management tool to the whole organization.*

Peter Senge the author of *The Fifth Discipline* [Senge94] highly emphasizes that a learning organization should know its systems. This is important if you want your employees to help you out “growing” your organization. An enterprise model can be used as:

- A “living” document that captures how the organization works.
- A map to discuss the current problems of the organization.
- A starting point to design and correct processes and organizations.
- As training material for new executives, managers, and staff employees that are new to the organization or that have changed roles.
- As a guideline or manual on how to perform a specific process.
- As a tool for measurement and accounting for organizations, processes, systems, applications, and networks.

In recent times the reasons for an enterprise model are becoming more and more compelling. As competition, change and customer expectations are increased a more “scientific” style management is emerging with equally important “hard” and “soft” issues. While it will always be important to listen to people, making them feel appreciated, set reasonable expectations, set clear career goals, provide constant education and well rounded benefit package, the harsh business environment also requires for a high degree of adaptability which requires the precision of detail business models. No longer can the enterprise be managed solely by using the leadership of its executives, nor can it just manage the numbers of the financiers, the revenue produced by the marketeers or the production of its operations; it must look at **ALL** of its processes and manage them collectively. However, it also true that one cannot dwell into every detail of the organization, a unique balance for a given organization must be found in order for the model to be simultaneously useful and manageable.

### **2.3 OO Zachman Framework**

How can the organization, its IS organization, its enterprise architecture and the enabling applications cooperate to deliver a conceptually integrated, congruent, coherent enterprise model within the constraints of BPR, object oriented technology and open systems and networks?

*The most widely used technique to perform enterprise models is the Zachman’s Framework [Zachman]. The one presented in table 1, is an object oriented-BPR version of the Zachman framework.*

This framework provides different views of the enterprise. The columns reflect unique models and the rows represent unique perspectives. No one column is more important than another and the columns and rows may be interrelated. Notice how the level of abstraction changes as we study the different rows. The first row is the executive view. The second row is that of business managers and business staff. Notice that eventually the organization merges with its applications through its enterprise system architecture to form a comprehensive whole.

Zachman's Models	Motivation	Participants	Activities	Products
Strategic Assessment	Generate an effective Organization	- Senior Management - Business Architect, etc.	Functional Strategy	- Case of Action - Vision Statement - Process Descriptions
Business Analysis, Design and Evolution	Generate effective process teams	-Process Owners -Reengineering Teams, etc. -Business Staff	Business Process Analysis, Design and Implement.	Detailed Enterprise Model, Design and Implementation
Information Systems Model	Generate effective enabling applications	- EW System Architect - Business Analysts,	System Analysis and Requirements	IS Architecture and applications' requirements
Technology Constrained Model	Generate an effective Enterprise Architecture	-EW Architect -Architecture Group, etc.	System Design	IS Architecture and applications' design
Detailed Representation Model	Generate effective application implementations	-EW Architect -Application Developers, etc.	System Evolution	IS Architecture and applications implementations'
Working System Model	Generate effective production systems to enable processes	Production Maintenance Groups, etc.	System Maintenance	Evolving Production Systems

Zachman's Models	Applicable Pattern Language	Tools in Focus	Risk Assessments	Techniques	Objects in Focus
Strategic Assessment	- PattBPR	-Financial Forecasts -Cost/Benefit Analysis	-Impact to whole organization	- Process textual descriptions	Large Scale Organization
Business Analysis, Design and Evolution	- PattBPR	-ABC tools -Process and project tools -CASE Tools	-Impact to specific process reengineering projects	-Business Use Cases - System Use Cases	- Small Scale Organization -BOM - Roles
Information Systems Model	-Large Scale Architectural Patterns [POSA96]	-CASE tools -Process and project tools	-Impact to application development	-Class Categories -Use Case Maps	Applications
Technology Constrained Model	-Business Patterns - Software Design Patterns [Gamma]	-CASE tools -Compilers -Debuggers - CM Tools	-Impact to overall System Architecture and applications	- Design Heuristics,	BOM -> DOM + mechanisms
Detailed Representation Model	-Idioms	-CASE tools -Compilers -Debuggers -Testing tools	-Impact to application development projects	-Edit, Compile, Link, Test	Object Code i.e. C++, Java, SmallTalk
Working System Model		-Performance Monitors -System Management tools	-Impact to production systems, support, maintenance	Executable Object Systems	Production Systems

**Table 1. Object Oriented Zachman Framework.**

( BOM = Business Object Model, DOM = Design Object Model, UCs = Use Cases, EW=Enterprise Wide, OOA = Object Oriented Analysis, OOD= Object Oriented Design, Implement.= Implementation)

*One very interesting feature of this version of the Zachman framework is that the number of distinct models required to model a complete organization including its organization, its IS organization and its enterprise system architecture is one: A comprehensive **Object Model**.*

### Correctness

Is a model correct just because it is made out of objects? Most definitely not. In fact correctness and object orientation are orthogonal.

*An enterprise model should be constructed with sets of proven **patterns, or best practices** at every level of abstraction.*

However, one cannot call something a **pattern**, a proven solution, until it has been used typically at least three times successfully.

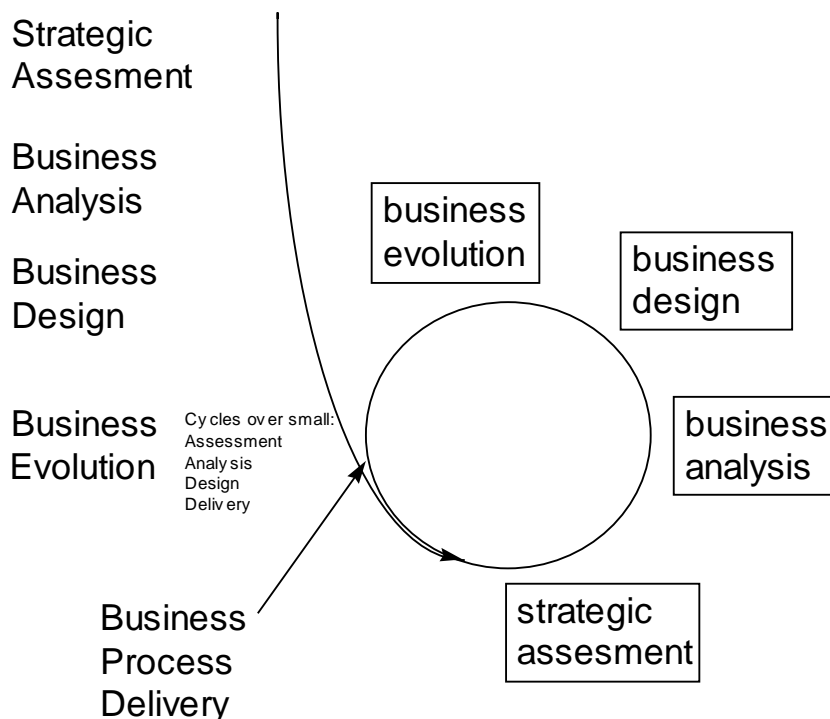
## 2.4 Business Engineering Method

It is impossible to create a generic project plan to reengineer a generic organization. There are just too many variables that need to be considered for example, political interactions, funding, human resources available and executive support to name a few. However, it is possible to create a template process that considers the bare minimum requirements of the stages of creating, evolving and implementing an enterprise model.

The business Organization evolves the enterprise and its model by iterating in a “b” like life cycle over the following steps, see figure 2:

1. Strategic Assessment
2. Business Analysis and Requirements
3. Business Design
4. Business Evolution

*To “generate” an enterprise using the PBR method constant iteration over each on of the rows in table 1 is required to generate, evolve and maintain the organization and its information systems through the release of stable baselines. This iteration takes place applying the PattBPR pattern language given below.*



**Figure 2. Iterative life-cycle for the business organization.**

There are many benefits of conducting this iterative and incremental life cycle, this argument is presented by many authors such as [Boehm81], [Booch95], [Brooks95], [Coplien95], [Sutherland96], and even [Hammer95] to name a few. In particular, the management of risks at every stage is ensured and a

working version of the processes is tested before going any further. Notice these steps are identical with those prescribed by an evolutionary software method such as the Booch method, except in this case it is the Organization and its processes the ones being delivered.

What is the result of these iterations? *Because we are using PROVEN patterns*: successful, adaptable, coherent, congruent, effective, efficient, flexible, resilient and scaleable organizations, IS organizations and system architectures. In reengineering terms, this translates into an organization with a shining **Business Systems Diamond**. The **Business System Diamond** states that *Business Processes* enabled by *Information Technology solutions* lead to *Jobs and Structures*, which in turn require *Management and Measurement Systems*, that reinforce a set of *Values and Beliefs*.

The method recommends finding most of the requirements of the business processes but only enough to create a business architecture for the organization. Then it maps the requirements into the design of the organization, but it only chooses to implement portions of this architecture in well chosen horizontal and vertical slices.

Who re-engineers? The business engineering process team. The organization has a process team that is fully dedicated to build enterprise components. In this sense the worker in this process team are “enterprise engineers” [MartinJ95]. The Business Architect and the enterprise engineers design the whole, that is, all the components of the OO Zachman framework including: strategic models, business models, enterprise architecture models, application models; and they implement releases of the enterprise architecture through releases. Why are the business processes redesigned outside the ongoing process? The people working in the process should concentrate in their work. Too many reengineering projects fail simply because the people that were responsible for reengineering were too busy working in the current process. In a sense is like trying to fix a busy highway. To get the job done you either stop traffic, which is not a viable business solution, or build the new highway in parallel - the best option if you can afford it.

#### **Summary of the Business Engineering Process**

The *Client Business Engineering Team*, develops new business processes including their enabling applications. The *Client Business Engineering Team* for a client develops a SINGLE comprehensive business model that has four inter-related parts. Because this team is reengineered, every process is centered around outcomes, so the four parts of the model have and associated sub-processes:

#### **Business Analysis leads to the Business Requirements Model**

A **Business Requirements Model** is built with **Business Use Cases** and **Business Process Scenarios** that is compatible with the reusable **Business Architecture Model**. The **Business Use Cases** simply define “interfaces” to do business in the form of inputs, outputs and task definitions. The **Business Process Scenarios** are representative cases of these interfaces that navigate through the **Business Architecture** model.

#### **Business Design leads to Business Process Model.**

A **Business Process Model** is created based on the **Business Requirements** and the **Business Architecture**. The **Business Process Model** leads to:

#### **Systems Analysis leads to Application Requirements Model**

An **Application Requirements Model** composed of **Application Use Cases** and **Application Scenarios** that satisfies the **Business Process Model** and **Business Architecture Model**. Similarly as defined above for the business design, the **Application Use Cases** define “interfaces” to the **System Architecture** and the **Application Scenarios** describe paths of execution thought the **System Architecture Model**.

**Application Design leads to the Application Design Model**

An **Application Design** that supports the **Application Requirements Model** and that is based in the reusable components of the **System Architecture Model**.

**Application Implementation leads to the Executable Model**

An **Executable Model** that satisfies the **Application Requirements Model** and the **System Architecture Model**.

**Business Evolution leads to Business Integrations and Business Releases**

The business engineering team builds the models described above using an iteratively and incremental life-cycle. The first integration is the product of the Business Design stage.

**Business Maintenance leads to Business Maintenance Releases**

The business engineering team continues the evolution of the releases business processes and applications after they have been released into production.

See figure 3 to see the structure of the Business Engineering organization.

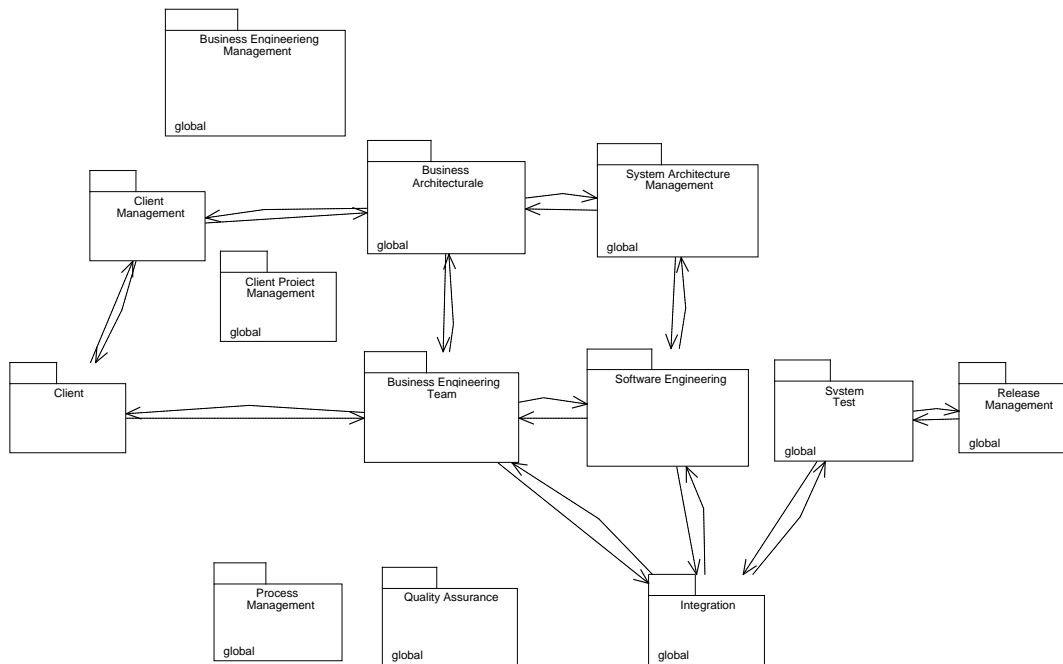
**2.5 Organization**

A behavioral organization can be modeled by a class diagram of actor “stereotypes”. The organization can be thought of as a collection of objects cooperating with each other. When control, ownership or delegation are modeled aggregation relationships may be used. When generalization relationships are modeled inheritance relationships should be used. When interaction is modeled association relationships are appropriate.

A diagram showing hundreds or thousands of classes could be confusing and would detract the architect from appreciating important high level concepts. Also, object oriented notation is not readily comprehensive to most audiences, particularly the executives and other members of the business organization. This comprehensive enterprise object model may be broken in highly cohesive and loosely coupled subsystems and layers of abstraction representing class categories.

*Class categories represent logical subsystems of clusters of classes and are a better tool to describe architectures in the large. The organization is partitioned into processes, and therefore Process Teams are created to support the business architecture.*

Figure 3 includes an example a class category diagram. This diagram uses the UML class category notation to identify process teams as logical and physical subsystems of the organization. In this example the structure of the business engineering organization is presented as an example.



**Figure 3. Business Engineering Process Team.**

This is a diagram anyone can understand, even a CEO. The arrows simply read, “depends on” in the direction of the arrows. Notice most of the class categories have arrow coming in and out. This is because people typically interact through bi-directional associations.

There is one very important realization one should make about this diagram:

*In any architecture there is a natural order of things in which one should proceed to achieve a coherent model by iteratively and incrementally building it. Dependencies in the enterprise model constrain its design and implementation.*

*When creating a project plan to write the enabling applications and subsystems of the enterprise wide architecture, it is critically important to map the dependencies of the business architecture and the enterprise system architecture correctly.*

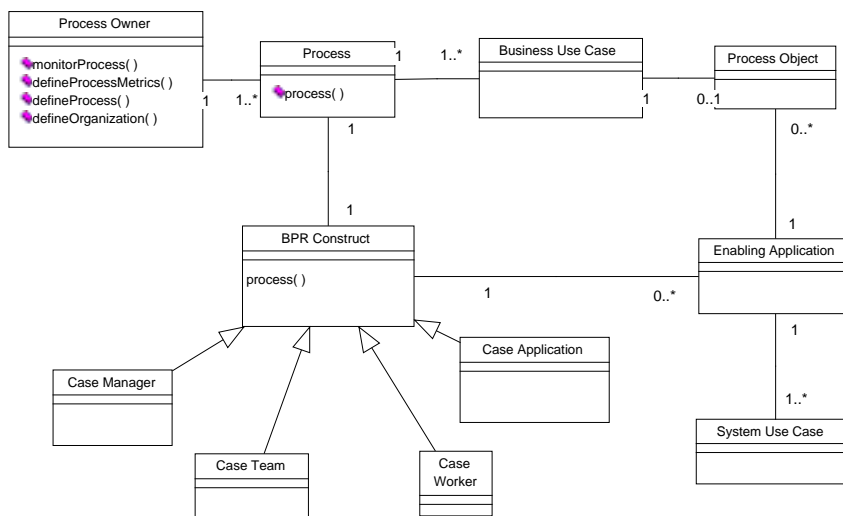
Ask a building architect if he would design and implement the plumbing system of a sky-scraper first and then the foundation. Or ask an electrical engineer if he would design a chip without finding the power limitations and fan-out constraints. Why then is it thought that an enterprise or a software architecture is any different. It is just a question of industry maturity.

Imagine the amount of human and financial resources wasted if an overall business architecture or the enterprise system architecture was flawed. The costs and risks for this “big bang” approach are prohibitive. This approach does not allow for any midcourse correcting mechanisms, it distracts the

organization because of its high interaction requirements with the operating organization and should be discouraged because it involves high costs and unmanaged high risks.

*From a dynamical system perspective this means reducing delays in a feedback system. The longer the time to make a “correction” in direction, the higher the uncertainty of the final state and the amount of work to be done.*

At a more detail level, each of the process teams will be partitioned in components using reengineering constructs such as a *Case Application*, a *Case Worker* or a *Case Team*, someone entirely responsible for a process, which in turn may use an enabling applications. This application will in turn use the enterprise system architecture subsystems to fulfill its requirements. See appendix B for a summary on the UML notation. It is well worth studying this notation since it is the most widely used and the fastest gaining worldwide adoption. Examples of detailed designs will be provided in the following sections.



**Figure 4. Organizational Framework of BPR constructs.**

Why do I say this is a framework? Because it is a reusable micro-architecture that allows one to create more specialized architectures. A process is designed with steps that are implemented through a reengineering construct. For example, to implement a process to “Get Loan” with a Case Worker we would do the following:

- 1) Create a derived Process Owner called “Loan Process Owner”. This process owner may own more than one Loan related process, such as monitoring a Loan status, or pay a Loan in full. However, we will concentrate in the “Get Loan” process.
- 2) Create a derived class from Process called “New Loan Process”.
- 3) Create as many Business Use Cases instances as needed which are realized by this process. See for example figure 4. For each business use case instantiate as many Process Objects as possible. This will give you a “high” enabling ratio.
- 4) Create a derived Case Worker called “New Loan Case Worker”.
- 5) Create a derived class of Enabling Application and associate it with the “New Loan Case Worker”.
- 6) Instantiate as many system use cases as needed for the enabling application.

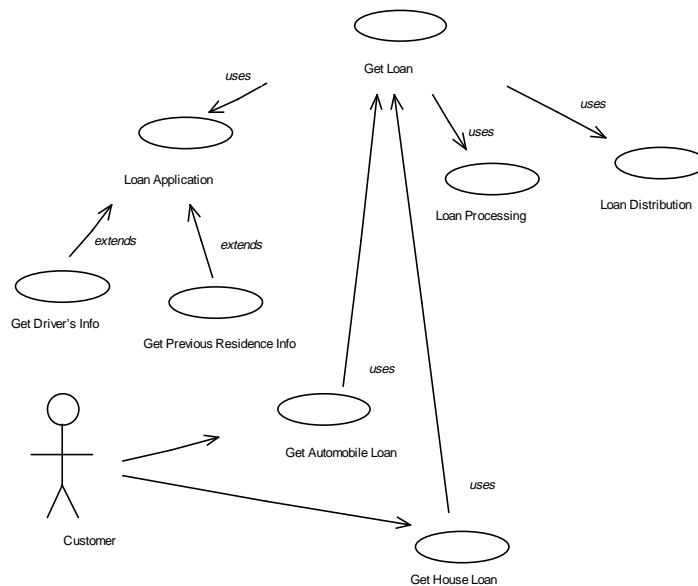
- 7) Define the “process” operation on the “New Loan Case Worker”. The “process operation” should use as many system use cases as possible from the defined enabling application. If at all possible discourage “manual processes”. However, this is not always possible.

What is the result? The result is that you have a well defined business process implemented by a Case Worker with its corresponding enabling application. And you have implemented the OMG's BAA. By compressing a business process that spanned functional silos in someone's company into an object, or "reifying" the process, we then manage the business process from within the software. This gives you the ability:

- 1) To edit the process defined in “New Loan Process” independently of its implementation.
- 2) Add process objects independently and therefore incrementally increment you enabling ratio. This also implies adding system use cases or possibly adding enabling applications.
- 3) To use BPR constructs as “Strategy patterns” [Gamma95] and to change the implementation of a business process at “runtime”. For example, in the outsourcing business it is possible to have a IVR system (Interactive Voice Response) system running the same process that a Case Worker implemented as a Benefits Service Representative.

## 2.6 Business Processes

A business process interface can be defined by a use case. A use case as defined by Jacobson, [Jacobson95] is: *A use case is a sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the system.*



**Figure 5. Use cases are the interfaces of a business process.**

Notice here that if we abstract the “reusable” use cases into a package. This package becomes a set of “reusable requirements” library providing reusability at the business requirements level and if enabling applications are defined for the processes then it implies “reusability” at the system requirements level.

An important consideration is to find a good template for the use cases and to ensure that they contain sufficient and appropriate information to describe the behavior. The release management process shown in the next section shows a template for a Business Use Case.

*Most problems in business and software development arise from the lack of good requirements. Eliminating "holes" at this stage saves considerable amounts of efforts in later stages.*

Once the use cases of the process are found, mapping of these use cases into the business architecture is done by means of use case maps (Also shown in figure 2). Once the process interfaces are found and described through use cases, reengineering patterns are used to reengineer the process. For example a process . All reengineering constructs compress an existing event trace that previously extended multiple functional silos.

*Case Applications, Case Workers, Case Teams and Case Manager structures complete a business process aided with information systems.*

Notice the difference with the traditional "assembly line", where work travels across functional specialists until is completed. These arrangement is very error prone, and it is highly susceptible for rework, iteration, batches, queues and many other systemic problems.

The following is an example of how a business process has been redesigned with a Case Team:

### **Release Management Process**

#### **Summary**

- 1) *The production environment is upgraded with a scheduled release of functionality by the release management process team and other supporting members of the organization.*
- 2) *The client is aware of the scheduled release and has a clear expectation of this process.*

#### **Pre-Conditions ( May also be viewed as INPUTS)**

##### **External Pre-Requisites**

- 1) *The CSRs (Customer Service Representatives) have been trained and posses documentation about the new business process release and its enabling application.*

##### **Commitment to Perform**

- 1) *A release management swat team is designated to be responsible for the delivery of the new functionality to the production environment.*

##### **Ability to Perform**

- 1) *A complete and documented release is available for the production environment.*
- 2) *Adequate resources and funding are provided for conducting the release management process.*
- 3) *The software managers, software engineers, and other supporting individuals involved in the release management process are trained in the procedures applicable to their areas of responsibility.*

#### **Organization**

##### **Process Owner**

*Implementation Manager*

##### **Supporting Organization**

*Integration Manager*

*Release Management Case Team Leader*

*Release Management Case Team:*

*Production Support Specialist*

*Database Administrator*

*Security Administrator*

*Database Administrator*

IVR Administrator  
 Testing Team

**Agents**

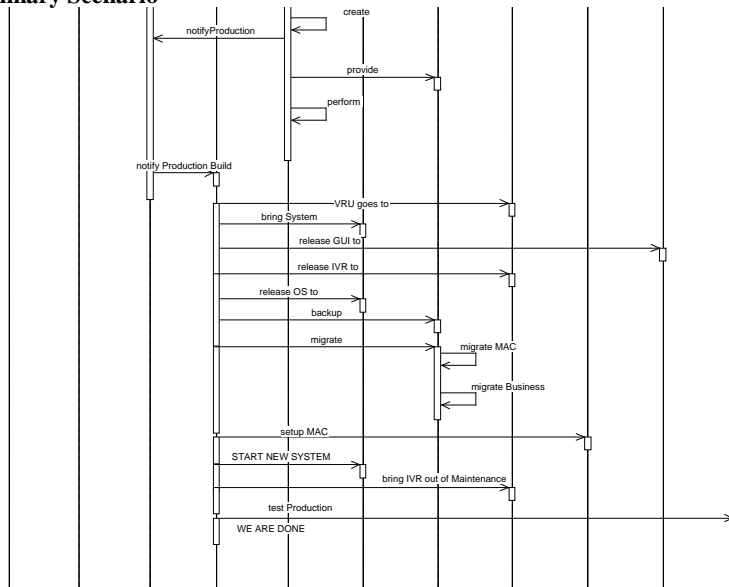
QA Testing Manager  
 Implementation Manager  
 Client Manager

**Process Description**

The Implementation Manager is notified when the QA Testing Manager signs off on a release. The Development Manager coordinates with other groups. The Development Manager then coordinates with the Client Manager the release time-frame and release window. The Development Manager also coordinates the production build with the Integration Manager. The Integration Manager creates the Production Release. The **Production Release Case Team** is assembled and a team captain is appointed. (This teams is an implementation of a **Case Team**). The Case Team releases the new release in the production environment.

**Scenarios**

**Primary Scenario**



**Figure 6. Primary scenario of the Release Management Process.**

**Process Metrics**

Release Schedule. Measurement of planned vs. actuals.

**Products**

Production environment. All executables, databases, scripts and other components.

System Documentation. All relevant system documentation such as User's Manual, Requirements, Architecture and Test Plans.

#### **Techniques**

Status meetings throughout the release schedule.

#### **Post-Conditions (May also be viewed as VALIDATED OUTPUTS)**

Either the new release is installed in the production environment in a stable configuration OR the old production environment is installed in production should any unrecoverable errors were found in the *Release Management Process*.

## **2.7 Applications as part of the Enterprise Model**

When the existence of an application is assumed in the design, some of the event traces of the business process will sink into the enabling applications defining their requirements. An enterprise may be viewed as a set of evolving real and virtual cooperating objects. Using objects to map enterprises facilitates enterprise design because human interfaces and applications can use the same notation. These *use cases* can be modeled as methods to the application thus defining the requirements for it.

As the level of reengineering increases the insertion of virtual objects increases. Also, object oriented methods provide a relatively easy-to-use and universal language closer to the business language already in use, therefore facilitating communications among functional specialists in computer technology and in business, and hopefully helping them to bridge the generalist gap.

## **2.8 Object Oriented architectures**

Object oriented architectures provide the best technical infrastructure for iterative reengineering. Business processes are supported by business concepts, i.e. business objects. An employee or a customer account are concepts that transcend a process and are reused across enabling applications. The applications use reusable server business objects. Business objects have now been promoted to first class citizens through OMG BAA [OMG-BAA95].

Developing application from a reusable architecture with "business objects" means having reusability in most areas of application development. However, most organization that practice 3-tier development with "business objects" expect to find it only in the middle tier. These are the reusable artifacts to be expected when application are developed from reusable components: (Notice all packages follow the DIP(Dependency Inversion Principle) with respect to abstract packages in figure 7.)

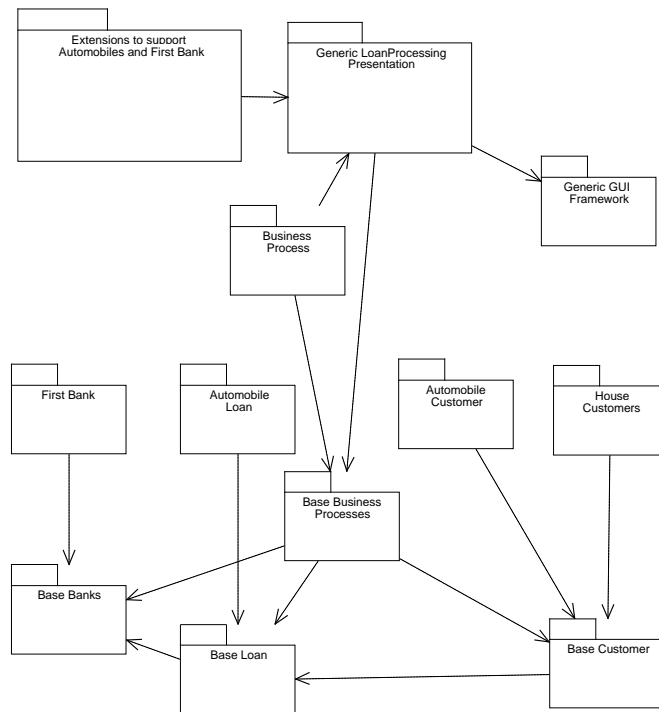
- 1) Reusable business model. That is a both the domain analysis and the business process.
- 2) Reusable business requirements as shown in figure 5, through use case relationships and business process objects as shown in figures 5 and 7.
- 3) Reusable presentations through frameworks in the presentation objects as shown in figure 7.
- 4) Reusable "process objects", the reification of the processes found in the business model. This is what "reengineers the process" giving you the Case Workers or Case Teams. By compressing the process in its reified format, a workflow manager can now "drive" units of work through multiple screens or multiple workstations.
- 5) Reusable "business objects" as shown in figure 7.
- 6) Reusable persistence. This gets very tricky.

From the process perspective imagine what that means. Having process owners in all aspects of the development process managing reusable resources.

For example, finding requirements from a reusable menu, through use case "trees" that map reusability in the requirements, having use case maps that clone each other representing parent-child relationships, providing reusability through frameworks using design patterns, cloned reusable testing banks for a family of use cases, cloned integration build based on a family of configuration management labels and branches, etc.

Also imagine what that does for the organization. A single architecture team managing an architecture that needs to provide for multiple clients, multiple Client Teams composed of Case Workers and Case Teams weaving threads through a reusable architecture, etc.

“Reusability” is everywhere and one needs to be prepared for it.



**Figure 7. Applications are generated from reusable architectures**

### ***2.9 Human Factors***

Other elements that the reengineering environment needs are proper hiring, education, employee benefits, measurement and management systems and bonus and compensation programs.

## 2.10 Simulations

Enterprise simulations can be used to understand what if scenarios. This is an important activity because what-if scenarios may be reviewed, assumptions may be tested and estimate of costs may be obtained as business design aids. This activity determines the boundaries for the potential return on investment of the reengineering project. An example of this kind of business simulation environments are the ones produced by: SES Software's **Business Architect** provide for OO BPR simulation environments also David Taylor's Enterprise Engines is finishing up Enterprise Engine™.

## 2.11 Universality

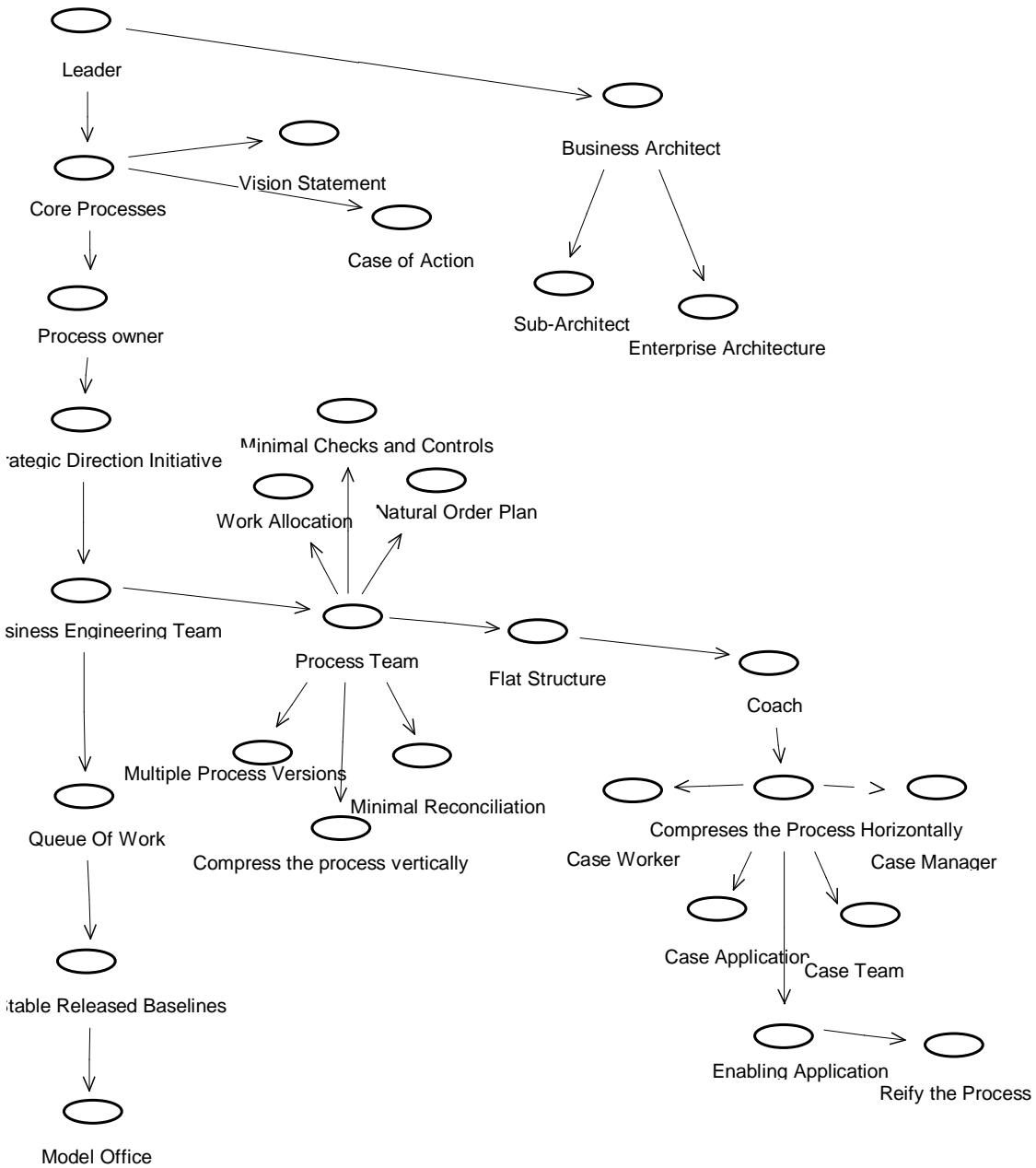
The techniques presented here are apply whenever business software is developed. It is possible to categorize business software according to the ownership of the software and the business process.

Mission-critical	The organization owns the business process and it also owns the software to support it.
Shrink-wrapped	The organizations owns the business process but it doesn't own the software to enable the process. This software sometimes is also referred to "licensed" software.
Outsourcing	The organization contracted out the business process and it doesn't own the software to enable it.
Contracting	The organization wrote the software but it contracted out the business process.

Notice there is always a central theme - business software *always* enables business processes.

*Regardless what composes the system architecture or who owns it, the same principles to generate, evolve and maintain the organization apply.*

## PattBPR- A pattern language for business process reengineering (Selected Sample patterns)



**Figure 8 PattBPR - A pattern language for business process reengineering.**

With the following global patterns.

**Global Patterns**

G1. Learning Organization, G2. Compensate Results, G3. Promote on Ability, G4. Productive Values

## ***Business Architect***

### **Alias**

Reengineering Czar [Hammer93]

### **Context**

To make decisions at the business strategy level the fundamental business architecture of the organization must be determined.

### **Problem**

The *Leader* is not someone that is an expert in process design, organizational structures, measurement and management systems or that has experience in change management issues. Who should be responsible for creating and documenting the existing and new business architectures? Who should be responsible to integrate all the sources of change into these models?

### **Forces**

Having a single individual controlling the conceptual integrity of the *Enterprise Architecture* of the organization can help in managing the conceptual integrity of the model but it may bias its architecture to those areas of interest or expertise of the architect and it may be a limiting factor in the speed of change. Having multiple inputs from many individuals can compromise the conceptual integrity of the model but it would allow for greater speed and diversity.

### **Solution**

The *Leader* appoints the *Business Architect* and gives him ownership of the *Enterprise Architecture* to ensure the its conceptual integrity. The *Business Architect* is an expert in business architecture and business process redesign. His primary responsibilities are helping the *Process Owners* to run each of their processes, overseeing the activities of the *Business Engineering Team*, controlling changes to the *Enterprise Architecture* to ensure that this is a coherent and congruent model of the organization, communicating to the *Leader* and the business *Sub-Architects*, and recommending resources to be assigned to the *Process Teams* and the *Business Engineering Team*. The *Business Architect* must have the support of the *Leader* so that his business designs and decisions are implemented. The headquarters of the *Business Architect* is the *Business Engineering Process Team*.

### **Resulting Context**

The *Enterprise Architecture* has an owner - the *Business Architect*, but the amount of work is too much to handle by a single individual.

### **Example**

### **Known Uses**

### **Pattern Connections**

#### **From**

*Leader*

#### **To**

*Enterprise Architecture, Sub-Architect*

## ***FlatReportingStructure***

### **Context**

Structures to and tactics to implement the *Process Teams* need to be chosen.

**Problem**

What is the best way to structure reporting hierarchies?

**Forces**

Organizations with long chains of command attempt to have fewer responsibilities for the participants in the management chain, this leads to a concentrated management practice; however, long reporting chains of command are slow to react and adapt to new business conditions and act as "Broken telephones" that convey incomplete or inaccurate information to upper level management layers.

Also long chains of command de-emphasize the ownership and involvement of the people that perform the work of the process; but short chain of command impose a higher responsibility standards for the members of the organization.

**Solution**

Create supporting organizations for *ProcessTeam* s that have flat structures.

The flatness of the organization is important to minimize cost, for accurate reporting to upper level management, for the prevention of middle-management empires and most important to shift responsibility and accountability for a business activity to the lower levels of structure such as CaseTeam, a CaseWorker or CaseManager s.

FlatReportingStructure means letting the workers of a process make decisions whenever possible., changing the worker's roles from controlled to empowered. However, care must be given to give the workers guidelines to exercise their new powers.

**Resulting Context**

Structures to implement flat organizations need to be chosen.

**Example**

The Catholic Church

**Known Uses**

Texas Instrument, IBM Credit.

**Pattern Connections****From**

*ProcessTeam*

**To**

*CoachCoach*

**Process Owners****Context**

The problems of the *Core Processes* are understood and it is necessary to define how these processes should look like in the future.

**Problem**

What is the best way to get accountability out of a process? Functional departments create assembly lines and the ownership of the process and its results are always in question.

**Forces**

Hiring, managing and building functional departments is easy; however, they create assembly line processes without encouraging ownership of the results for the customers. Allocating ownership to processes has more customer focus, because the products for a client have more accountability.

**Solution**

The *Leader* assigns *Process Owners* corresponding to the *Core Processes* of the organization. Good candidates to become *Process Owners* are old functional managers which understand the importance of the new focus on lateral management.

These *Process Owners* are sometimes already realizing most of the work in their functional management positions but haven't been officially assigned the ownership of the process. Hiring external *Process Owners* will send a strong signal to the organization in case there is opposition by the current functional managers.

In some cases, when the process is very broken *Process Owners* play the role of "lateral" managers and coordinate activities across large functional units, at least temporarily while *Process Teams* are created.

One could say that almost anything could be a CoreProcess; however the key is to concentrate in complete business activities that produce outcomes for customers.

**Resulting Context**

Structures, tactics and priorities to redesign the *Core Process* and the system architecture need to be chosen.

**Example**

IBM Credit's Wayne Hooever, Regis Filtz, Bell Atlantic's CAS Leader.

**Known Uses**

IBM Credit, GTE, Aethna Life, TI Semiconductor Group

**Pattern Connections****From**

*Case of Action*

**To**

*Strategic Direction Initiative*

***CompressTheProcessHorizontally*****Context**

Characteristics of the structures that perform the actual work need to be determined.

**Problem**

How should the structures that provide work should be architected?

**Forces**

Creating organizations of "functional specialists" that pass work among each other was the philosophy of the Division of Work that came out of Adam Smith's *The Wealth of Nations*. However, these organizations discourage the view of the customer and create systemic problems such as queues, batches, feedback loops and delays.

People in general tend to be specialists, so hiring and training a functional organization is easy. Instead training someone to accomplish the work of an entire business process could result in unrealistic expectations.

### **Solution**

Combine several jobs into one whenever possible using the CaseWorker, CaseTeam, CaseManager and CaseApplication organizational patterns. These are organizational structures that assign a single organizational construct to be responsible for a whole process instance. A consequence of this is that jobs change from single task to multi-dimensional work, but in doing so many "systemic" problems such as queues, batches, feedback loops and delays are eliminated.

CaseWorker s, CaseTeam s, CaseManager s and CaseApplication s, always  
CompressTheProcessHorizontally.

### **Resulting Context**

The structures that perform the work compressing the business processes need to be determined.

### **Example**

#### **Known Uses**

Developer Controls Process [Coplien95], Tall-fat men [Booch94], [Hammer93], [Senge90], [Hammer96],

#### **Pattern Connections**

##### **From**

*Reengineering Team - Business Engineering Process Team*

##### **To**

*Case Application, Case Worker, Case Team, Case Manager*

## **Case Team**

### **Context**

The structures that perform the work compressing the business processes need to be determined.

### **Problem**

The realization of a business process may take several forms. Are there organizational structures that enhance productivity, cooperation, customer focus and at the same time induce the growth of the company and the participants of the process?

Assembly lines create "systemic" problems such as queues, batches, feedback loops, delays, high inventories and high ratios of checking and control activities vs. value added activities. High "inventory" in creative or intensive human labor environments translate in large amount of idle human resources. What is the best way to organize resources that are capable of fulfilling all the needs of a given process?

### **Forces**

Hiring and managing "specialists" is relatively easy and it is well understood; however, this leads to assembly lines with "systemic" problems, and to unhappy employees whose potential and growth is constrained by their own specialization.

Hiring, mentoring and building teams of cross-functional resources is challenging; however, this leads to efficient processes and for desirable interactions among the members of the team that allow them to grow as individuals.

### **Solution**

Create a *Case Team* composed of cross functional resources that can fulfill the needs of the whole process. The “mix” of a *Case Team* needs to be carefully evaluated in order to make the team whole.

The members in a *Case Team* have "fuzzy" ownerships and responsibilities regarding the execution of tasks and artifacts within the process. As a consequence of their constant interaction, the members of a *Case Team* constantly learn from each other and have a better chance to enjoy a dynamic working environment. This gives them a "growth" path by learning from more experienced or more knowledgeable members in the *Case Team*.

They are also capable to reorganize themselves and adapt to multiple business situations and environments and therefore they are a good tool to harness “uncontrollable” processes. Their diversity gives them a chance to use their individual strengths in different situations, without being limited or confined to a specific role forever.

Finally, it also allows them to exercise their leadership abilities periodically without being exposed to larger responsibilities and the pressure of an assigned position of responsibility. This gives them a chance to grow as leaders and assume greater increased responsibilities as time evolves. *Case Teams* always have *Enabling Applications* to help them accomplish their duties.

The *Case Team* is bounded by the task that it is assigned to, it emphasizes "fuzzy" internal ownership, but "strong" external ownership, it uses a SCRUM like approach to develop deliverables [Sutherland96] and it works on the basis of *Encourage Productive Values* among its members.

### **Resulting Context**

Specific tactics to implement the design of the EnablingApplication need to be determined.

### **Example**

#### **Business Environment**

IBM Credit’s cross functional teams.

#### **Application Development**

Replacing the Developer role in *Developer Controls Process* with a cross functional team of developers in a three-tiered client-server architecture development effort using *Form Follows Function*.

#### **Known Uses**

Ford, GTE, Aethna, Hallmark, PepsiCo and most other Global 2000 companies.

#### **Pattern Connections**

##### **From**

*Compress the Process Horizontally*

##### **To**

*Enabling Application*

##### **Alternate**

*Case Application, Case Worker, Case Team, Case Manager*

## ***Minimal Checks and Controls***

### **Context**

The context is the work environment of large organizations where processes with checks exist.

### **Problem**

A large amount of checks and controls slows down processes and increase overhead for an organization.

### **Forces**

Checks are needed to ensure quality; however, too many checks and controls reduce productivity; however, having no checks or controls at all leads to poor quality products and services and unmanageable organizations.

### **Solution**

Minimize in as much as possible the checks and controls imposed into a process. Keep only the checks and controls that are absolutely necessary. Implement checks and controls in the natural breakdown of work.

### **Resulting Context**

### **Example**

#### **Software Development**

Make software "visible" by having the models produced by the members of the software organization always available online, this will provide with constant quality checks of the ongoing work. Only review these documents in a formal way at critical stages of the development process such as the end of analysis, design, iterations, etc.

### **Known Uses**

Insurance Companies (small claims), Banks (cash station and credit card small claims fraud); [Hammer93]

### **Pattern Connections**

#### **From**

*Flat Structure*

#### **To**

*Coach*

## ***Minimal Reconciliation***

### **Context**

The context is the work environment of large organizations where there are many inter-related "documents" about the same subject.

### **Problem**

Reconciliation across many artifacts/documents of related information is difficult and requires large amounts of work.

### **Forces**

Documenting a solution in more than one way is beneficial to the understanding of the problem; however, it creates the problem of reconciliation this related information. Documenting once a subject has the

benefit of requiring minimal reconciliation; however, it may require different audiences to understand the same level of documentation.

**Solution**

Minimize the number of by-products or artifacts that a project uses. The less documents of artifacts there are to reconcile the less amount of overhead work there will be.

**Resulting Context****Example****Software Development**

Keep as fewer artifacts as possible, for example versioned copies of: the requirements document, the architecture document and design documents, the code, the testing plan, the project plan, and the process and organization document. Even if these documents require nesting or linking with other documents avoid duplication of information that needs to be reconciliated.

**Known Uses**

Wal-mart/P&G/Other vendors; Booch minimal documentation set for a project [Booch95]; [Hammer93]

**Pattern Connections****From**

*Flat Structure*

**To**

*Coach*

## References

- [Alexander79] C. Alexander. *The Timeless Way of Building*, Oxford University Press, New York, 1979.
- [Beedle95] M. Beedle, *Object Based Reengineering*, Object Magazine 4(2), 1995.
- [Boehm81], B. Boehm, *Software Engineering Economics*, Prentice Hall, Upper Saddle River (NJ), 1981.
- [Booch95] G. Booch, *Object Solutions*. Addison and Wesley, Reading, 1995.
- [Brooks95] F. Brooks, *The mythical man-month*, Addison and Wesley, Reading, 1995.
- [Coplien95] J. Coplien and D. Schmidt, *Pattern Languages of Program Design (A Generative Development-Process Pattern Language)*, Addison and Wesley, Reading, 1995.
- [Hammer93] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Collins, New York, 1993.
- [Gamma95] E. Gamma et al. *Design Patterns - Elements of Reusable Object Oriented Software*. Reading, MA. Addison-Wesley, 1995.
- [Hammer95] M. Hammer and S. Stanton, *The Reengineering Revolution*, Harper Collins, New York, 1995.
- [Jacobson95] I. Jacobson et al. *The Object Advantage*. Reading, MA. Addison-Wesley. New York, 1995.
- [MartinR95] R Martin, *Designing Object Oriented C++ Applications using the Booch Method*, Prentice Hall, Englewood Cliffs, 1995.
- [MartinJ95] J. Martin, *The Great Transition*, AmaCom, New York, 1995.
- [OMG-BAA95] OMG, *OMG Business Application Architecture*, White Paper, Framingham, MA, 1995.
- [Senge94] P. Senge, *The Fifth Discipline - The art and Practice of the Learning Organization*. Benjamin Cummings, New York, 1994.
- [Shelton-F] R. Shelton, *Oobe: The Modeler's Handbook*, Addison and Wesley, Forthcoming title.
- [Sutherland96] J. Sutherland, *Scrum Web Home Page*, <http://www.tiac.net/users/jsuth/scrum/>, 1996.
- [Taylor95] D. Taylor. *Object Oriented Business Engineering*. Englewood Cliffs, New Jersey. Prentice Hall, 1995.
- [UML96] G. Booch, J. Rumbaugh, I. Jacobson, *Unified Method*, version .91, Rational, 1996.
- [Yourdon95] E. Yourdon, P. Harmon, *BPR and Software Development*, Cutter Information Corp., Arlington (Ma ), 1995.
- [Zachman] J. Zachman, *A framework for information system architecture*, IBM Systems Journal, 26(3), 276-92.